

Moving Target Techniques: Leveraging Uncertainty for Cyber Defense

Hamed Okhravi, Kevin S. Bauer, and William W. Strelein

Cyber moving target techniques include randomizing cyber system components to reduce the likelihood of a successful attack, adding dynamics to a system to shorten the lifetime of an attack, and diversifying otherwise homogeneous collections of systems to limit the damage of a large-scale attack. A review of five dominant categories of cyber moving target techniques assesses their benefits and weaknesses.

Securing critical computer systems against cyber attacks is a continual struggle for system managers. Attackers often need only find one vulnerability (a flaw or bug that an attacker can exploit to penetrate or disrupt a system) to successfully compromise systems. Defenders, however, have the technically difficult task of discovering and fixing every vulnerability in a complex system, which usually comprises an operating system, device drivers, numerous software applications, and hardware components. Within the cyberspace, this imbalance between a simple, one-bug attack tactic and a complicated, multipart defense strategy favors attackers. While defensive applications have grown significantly in complexity and size over many years, malicious software, i.e., malware, has remained relatively simple, computationally small, and still effective in bypassing defensive applications [1].

A major contributing factor to the imbalanced security of the cyberspace is the static nature of systems and defenses. The same copy of a popular software application with the same internals developed by a major software vendor may run on millions of machines. As a result, an attack designed to infect that software application is likely to compromise millions of machines. Similarly, many defensive applications are static; they discover suspicious inputs by applying a set of rules and checks commonly used by software built to detect attacks. Therefore, clever cyber invaders can craft attacks to bypass existing defenses by analyzing local copies of readily available defensive applications and then exploiting the weaknesses within those applications.

A promising approach to defense that attempts to rebalance the cyber landscape is known as cyber moving target (MT) defense (or simply moving target). Moving target techniques change the static nature of computer systems to increase both the difficulty and the costs (in effort, time, and resources) of mounting attacks. Colloquially put, these techniques turn systems into moving targets that will be hard for cyber villains to hit. Defenders using MT techniques pursue any or all of the following goals: make computer systems more dynamic by changing their properties over time, make internals of computer systems more random and nondeterministic, and make computer systems more diverse.

Although numerous techniques categorized as MT ones have been offered in the academic literature, we are limiting our overview of dynamic MT techniques to those in five computer domains – platforms, runtime environment, software, data, and network. Readers can find a more detailed discussion of these five categories of MT techniques in Okhravi et al. [2].

Moving Target Overview

An overview of different components of a computer system is a helpful start to understanding the domains of MT techniques. For ease of design and implementation, a computer system (e.g., a desktop or laptop machine, a mobile device, or a process control machine in an industrial control system) often consists of multiple layers of software and hardware. These layers are commonly referred to as the software stack although the stack includes the hardware elements as well. Each layer relies on other layers for its proper operation and function. Figure 1 presents one representation of such a layered design. At the very bottom of the software stack are the hardware components of the machine: the processor, the motherboard, the memory cards, and other peripheral devices and cards, such as a sound card and video card. Above this layer resides the operating system, which is responsible for controlling and managing the hardware components and providing an abstraction of them to the application. This abstraction is key to the interoperability and compatibility of the applications because the vast majority of the applications do not interact directly with the hardware components; rather, they use the operating system's abstraction. The abstraction layer, which is the interface that the operating system provides to the application, is sometimes referred to as the runtime environment. The hardware and operating system of a machine are collectively called the platform. Above the operating system reside the applications that are used to process and present data. The data themselves and their representation can be considered a layer atop the application. Finally, many systems do not operate as isolated devices but, in fact, are connected to other machines through a network. In general, five domains of MT techniques address dynamically changing the abovementioned software stack layers.

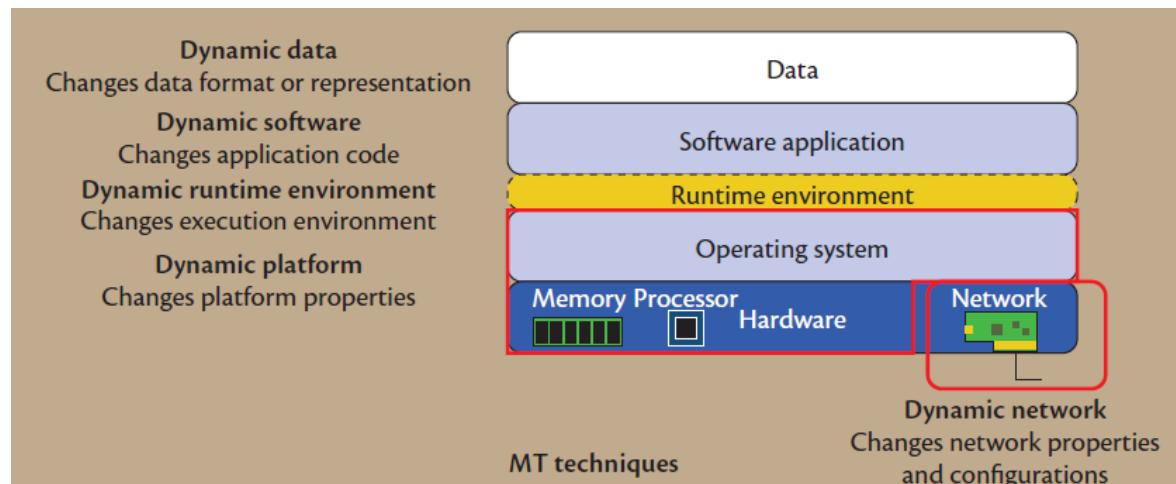


FIGURE 1. On the right is a depiction of the software stack. The layers of the stack address the five different domains of cyber moving target techniques discussed in this article.

Dynamic Platform

The dynamic platform domain consists of cyber defensive techniques that dynamically change the properties of the computing platform. Consider a system that runs a given application on top of multiple operating systems and hardware architectures. The application can run on top of a platform consisting of the Fedora operating system and x86 processor architecture or a platform consisting of the FreeBSD operating system and ARM processor architecture. Such a system can be implemented by compiling the application to different processor architectures and implementing a platform-independent checkpointing mechanism to preserve the current state of the application during platform changes [3]. This type of system illustrates a dynamic platform MT technique. Other examples of dynamic platform techniques include a voting system that runs an application on top of different platforms, each platform voting on the output of the system [4], or a system that randomizes the operating system's internals that are unimportant for the correct functionality of the application.

The major benefit of a dynamic platform technique is that it can prevent platform-dependent attacks. Crafting a successful exploit against a system usually requires that an attacker consider the exact platform of that system. By varying the computing platform, an MT technique can mitigate attacks that are platform-dependent. An attacker can develop a strong attack by incorporating different exploits against different platforms, but this approach increases the costs (in time, computation complexity) of developing the attack. Note that dynamic platform techniques cannot mitigate attacks that target a higher-level application logic flaw and that do not depend on the platform. For example, SQL¹ injection attacks, which inject malicious commands into a database application by leveraging a flaw in the application's high-level logic, are typically not mitigated by dynamic platform techniques.

While dynamic platform MT techniques offer the potential to defeat platform-dependent attacks, these techniques can increase the complexity of the overall system, are generally difficult to effectively manage, and can actually be detrimental to security if used inappropriately [5]. Perhaps the greatest challenge from a system complexity and management perspective is the synchronization of application state across the set of diverse platforms. Examples of application states could include open data files, user input from a keyboard or mouse, or network traffic that needs to be correctly delivered to a specific running process (while correctly maintaining connection-specific state in the kernel). Synchronizing these resources among the dynamic platforms in real time requires a complex management infrastructure that can migrate state with speed and agility. Such a management infrastructure increases system complexity considerably.

Another potential limitation of dynamic platform techniques is that the use of multiple distinct platforms can actually increase the system's attack surface, that is, the components of the system that are exposed to and could be targeted by a potential attacker. Suppose that a dynamic platform MT technique migrates an application between three platforms: Linux,

¹ SQL stands for Structured Query Language, a standardized programming language for requesting information from a database.

Windows, and Mac. If the attacker has an exploit that works on the Windows host, the attacker simply needs to wait until the application migrates to the Window host to launch the exploit and compromise the application. Making the program migration less predictable can help, provided that the attacker cannot reliably guess which platform is running the application.

Dynamic platform techniques are only effective defenses when the attacker must compromise all platforms (i.e., called an *in series* configuration) not just one platform (i.e., an *in parallel* configuration). If the attack requires a long time to succeed (a long-duration disruption of service), a dynamic platform approach can be helpful to thwarting the attack; for short-duration attacks, that approach can be detrimental to security because the attacker's goal may be accomplished on one platform.

Dynamic Runtime Environment

Techniques in the dynamic runtime environment domain dynamically change or randomize the abstraction provided by the operating system to the applications, but without hindering any important functions of the system. One of the most important abstractions in a computer system is memory. For various reasons, including isolation of different applications, compatibility, and interoperability, a memory location that is presented to an application in most modern computer systems is not a direct representation of the actual physical memory. Rather, a redirection is applied by the operating system, i.e., an abstraction known as the virtual memory. A well-known dynamic runtime environment MT technique randomizes what addresses in the virtual memory are used by the application. The technique is typically referred to as address space layout randomization (ASLR) [6] and is implemented in most modern operating systems including Linux, Windows, Mac OSX, Android, and iOS. By randomizing the addresses, ASLR makes exploit development significantly more difficult for attackers because attackers do not know where to place their malicious code. Other dynamic runtime environment techniques include those that change the processor instruction encoding (also called instruction set randomization), or finer-grained variants of ASLR in which smaller regions of memory are randomized.

Dynamic runtime environments are among the most practical and widely deployed MT techniques. Despite the success of this MT domain, two important weaknesses can allow an attacker to circumvent the defense. First, ASLR requires memory secrecy. If the contents of memory are disclosed or leaked to an attacker, the attacker may be able to use this information to defeat ASLR. Such memory disclosures are possible via separate vulnerabilities, known as buffer over-read vulnerabilities, in which the contents of memory are read beyond the allowed boundary, disclosing how memory has been randomized. Without strict memory secrecy, an attacker can circumvent the protections provided by ASLR to launch code injection or code reuse attacks. Second, the low granularity of randomization in many ASLR implementations reduces the overall protection provided by the technique. For example, in Linux, only the start location of certain memory regions (e.g., dynamically linked libraries) is randomized by default, and the executable program code itself is often not compiled with ASLR support. As such, this section of the program's memory is not protected and can be a vector for exploitation.

Dynamic Software

In the dynamic software domain, MT techniques randomize or diversify the internals of the software application. One technique, the multicompiler [7], creates from the same source code (e.g., written in C) different versions of software executables (binaries) that perform the same function but are different internally. The different versions can arise from various processor instructions used during the compilation process or from the same instructions but used in different locations inside the executable. Note that a given copy of the executable with a given set of internals may never change, but different machines in an enterprise may run different executables. In other words, this technique can create spatial diversity (i.e., diversity among many machines) as opposed to temporal diversity (i.e., diversity in one machine over time). The major benefit of dynamic software techniques is that they mitigate the impact of large-scale attacks. If an exploit is designed against a given variant of the executable, that exploit will have a small chance of working against other variants of the executable. Hence, an attacker cannot compromise many machines at once. This situation is contrary to the current one in which an attacker develops malware that can successfully compromise many machines running the same target application. In recent sophisticated breaches, attackers reuse parts of the benign code of the target application itself to achieve malicious behavior. Known as code reuse attacks, or return-oriented programming attacks [8], these attacks can successfully circumvent existing defenses that detect and stop foreign pieces of code. By varying the benign application code, dynamic software techniques can effectively foil code reuse attacks.

Dynamic software techniques often use specialized compiler techniques to produce executable software variants with different and unpredictable memory layouts. These variants may use padding (adding meaningless bytes of data) to make the size of memory regions unpredictable. They also may contain within the executable code a no-operation (NOP) instruction that does not perform any operation but can make code reuse attacks hard to launch because the instruction changes the location of other instructions.

Dynamic software techniques suffer from a variety of weaknesses. Recompilation to produce a software variant requires access to a program's source code and is not possible with proprietary, third-party software for which source code is not made available. Furthermore, ensuring correct operation of the compiled variant can be challenging because one cannot simply verify a known integrity measurement of the executable file to guarantee that the code has not been (maliciously) modified.

Another drawback of dynamic software methods is that software is often compiled with special optimization flags that reduce the space and/or computational complexity of the compiled binary code. Utilizing an MT technique that explicitly compiles the software to introduce randomness in the memory layout (by randomizing the size and/or location of objects) may not be compatible with the space saving or compute-time saving optimization passes performed by the compiler. Consequently, the dynamic software is unlikely to maintain the same performance properties as the ideally optimized compiled code.

In addition, dynamic software techniques that use execution monitors to instrument and compare multiple versions of an executable introduce significant performance costs. For example, if an MT technique compares execution for an application that has two variants, there is at least a twofold performance cost relative to native execution of the application (in terms of processor, memory, and input/output utilization). This cost may be reasonable for protecting one or two applications for which the highest degree of security is required, but the cost is likely unacceptable for the protection of all applications running on a host. Techniques in the dynamic software domain may also be subverted by information leakage attacks. If attackers can expose how an executable has been diversified, they can attack it as if it were not diversified at all.

Dynamic Data

Moving target techniques under the dynamic data domain change the format, syntax, representation, or encoding of the application data to make attacks more difficult. In this domain, the diversity can be temporal or spatial. For example, to protect a Linux operating system, a defender could dynamically change the representation of the user identifier (UID) that determines what access rights a user has. This defense is effective against attempts that seek to increase a user's access rights; for example, an attacker may try changing the UID to that of an administrator so that the attacker can exploit the expanded rights to access sensitive resources. This type of attack is one example of a larger class known as privilege escalation actions that can be mitigated by UID randomization.

Dynamic data techniques offer the promise of protecting data from theft or unauthorized modification, but these techniques suffer from two important weaknesses. First, the number of acceptable data encodings is limited. For example, for encoding binary data either the base64 or the hexadecimal encoding scheme would most likely be used because there are few other accepted standards for data encoding. Nonstandardized schemes are certainly possible, but these may increase the complexity of the interoperation among system components. Second, the use of additional data encodings may also increase the attack surface of the software. For each encoding type, the software must have the proper parsing code to encode and decode the data. This additional parsing code itself could have security-relevant software bugs.

Dynamic Network

Techniques in the dynamic network domain change the properties of the network to complicate network-based attacks. One such technique frequently changes the Internet protocol (IP) addresses of the machines in an enterprise network [9]. This IP rotation technique can thwart rapidly propagating worms that use a fixed hit list of IP addresses to infect a network. Another technique, known as an overlay network, creates dynamically changing encrypted tunnels (i.e., encrypted communication connections over public networks).

Dynamic networks is an appealing class of techniques to reduce an adversary's ability to conduct reconnaissance on a network, map a defended network, or select specific hosts for a targeted attack. However, these techniques face two important obstacles to deployment. First, because many dynamic network techniques lack a well-articulated threat model, it may be

unclear to network defenders what threat needs to be mitigated. Consider a technique that isolates a small group of machines from the larger network (or Internet). If hosts within the isolated network can still communicate to hosts beyond the isolated network, protected hosts may be vulnerable to any number of client-side attacks that exploit vulnerabilities within the unprotected hosts' web browsers or document viewers. For example, targeted spear phishing (fraudulent email messages that try to elicit information such as passwords to Internet accounts) could penetrate a protected network through its connections to unprotected hosts. Dynamic network based MT techniques do not address these types of attacks.

Second, many dynamic network techniques introduce randomization into the fundamental protocols that are used on the Internet. However, the effectiveness of this randomization at foiling attacks is unclear. Suppose an MT technique randomizes network identifiers (such as the IP address). If service discovery protocols such as the domain name service (DNS) are used to convert human-readable domain names to machine-readable IP addresses, these services may undo any potential security benefit obtained through the MT technique itself, provided that the attacker can issue DNS queries.

Summary

One way to understand the benefits of MT techniques is to look at the steps of a cyber attack that these techniques are trying to mitigate. To successfully compromise a system, an attacker must progress through the several phases depicted in Table 1. The first phase is conducting reconnaissance; an attacker collects information about the target. The second step is accessing the victim; the attacker collects enough information about the configurations, applications, and software versions that are running on the target machine to develop an attack against it. During the third stage, the attacker develops an exploit against a vulnerability in the target machine. Next is the launch of the attack, which may include, for example, sending a malicious network packet to the target machine, luring the user to click on a maliciously crafted link, or using a malicious thumb drive. After the attack is launched and verified, the attacker may take additional steps to maintain a foothold on the target machine (i.e., persistence). These phases, together referred to as the cyber kill chain, are correlated in Table 1 with the MT technique domain that is aimed at mitigating the effectiveness of each step.

Table 1. Primary attack phases disrupted by techniques in the five domains.

MT domains	Attack phases				
	Reconnaissance	Access	Development	Launch	Persistence
Dynamic networks	✓			✓	
Dynamic platforms		✓	✓		✓
Dynamic runtime environments			✓	✓	

Dynamic software			✓	✓	
Dynamic data			✓	✓	

Effectiveness of Moving Target Techniques

Understanding the weaknesses of existing MT techniques has motivated Lincoln Laboratory researchers to develop criteria for evaluating the techniques' effectiveness. By studying attacks against well-known MT techniques, we identified three major problems that contribute to the weaknesses of such techniques. First, in some cases, the dynamic change in the system is too slow, allowing an attacker to observe the current state of the system through, for example, information leakage attacks; to craft an attack against the current state; and then to compromise the system by launching the attack, all within the interval between two system changes. Second, the space of the dynamic change may be too small. In a system that has two possible states, attackers may not know the current state of the system, but they will have a 50% chance of success in attacking the system purely by guessing the current state. In many MT techniques, attackers can also reduce the amount of uncertainty they are facing by quickly testing every possibility; this strategy is known as the brute-force attack. Third, in some MT techniques, elements of an attack surface are dynamic (e.g., linked libraries), whereas other elements remain static (e.g., executable program code). The static elements become a target of attack because they do not present any uncertainty for the attacker.

Using the above insights, we determined three criteria for evaluating an MT technique – timeliness, unpredictability, and coverage:

- Timeliness is defined as the extent to which a defender's movement can be applied between the time at which an attacker makes an observation and the time at which an attack is completed. Timeliness evaluates how fast the system moves. Because the actual time between movements depends on the attack model of concern, the definition of timeliness includes the possible attacker's observations. In fact, an optimal MT technique should tie movement events to possible actions that can leak information to an attacker.
- Unpredictability is the extent to which the outcome of current or future changes to the attack surface are indeterminable by an attacker. Unpredictability evaluates the uncertainty faced by an attacker. A quantitative metric for unpredictability is information theoretic entropy.
- Coverage is the extent to which all elements of a defended attack surface are subject to change. Coverage evaluates whether or not the MT technique changes every element of an attack surface. If some parts of the attack surface remain static, they can become the target of attacks.

We have also developed rigorous, quantitative metrics for these criteria [10], the discussion of which is beyond the scope of this article. We use these metrics to evaluate MT techniques and analyze the protection they provide against cyber attacks.

Practical Considerations

When deciding to deploy an MT technique, system defenders have many practical issues to consider. They should understand the potential impact of the MT technique on the system's performance. Many MT techniques offer security against strong adversaries, but incur performance penalties that for some applications could be prohibitively high. Recognizing the performance requirements of the system and the expected performance costs of the MT technique can help defenders make the right decision about deploying MT defenses.

Defenders should also understand the effectiveness of an MT technique against a relevant threat model before it is deployed. Techniques that have shown high effectiveness against realistic attack models should be selected before those that have uncertain benefits or those that protect against an unrealistic threat. Hence, it is important to have access to a well-defined attack model that describes the exact types of attacks that are of concern and that are relevant to the system being protected.

Finally, MT techniques do not necessarily solve all security problems; rather, they are best suited to defending against specific threats. Defenders, therefore, should understand the composability (i.e., combinatorial possibilities) of MT and non-MT techniques so that they can enhance protections against cyber security threats. For example, defenders may want to guard against code injection attacks by using ASLR. But to improve security even more, they could add signature-based network monitoring to examine network traffic in real time and drop all packets that appear to contain code injection payloads.

Future Directions

Future research in MT techniques will take multiple directions. In designing new techniques or evaluating existing ones, researchers should analyze whether or not the additional complexity created by the randomization or diversification of the system's components is actually exposed to a potential attacker. Many MT techniques create complexity in a system component, but attackers can avoid or bypass the complexity through attacks that exploit information leakages or attacks that work regardless of the specific internals of a component (e.g., higher-level logic flaws in the application). The challenge for defenders, then, is to ensure that the complexity is not exposed to the system's operators and maintainers. Ease of deployment, operation, and maintenance is important for widespread deployment of cyber defensive techniques.

Additional work is needed in the area of evaluation and assessment of MT techniques. For cyber security to transition from a craft to a science, it is important for researchers to have concrete, meaningful, and repeatable evaluation methods. An imperative part of evaluation is the development of metrics that define measurement units of security and that can be used to evaluate the absolute security offered by an MT technique and a comparative assessment of it against other techniques. Meaningful and objective evaluation of MT techniques can benefit

from a variety of evaluation approaches, including abstract analysis, modeling and simulation, test bed experimentation, and real-world measurements in operational systems.

Finally, an important future direction for MT research is the examination, study, and evaluation of the composability of MT techniques with other MT and non-MT defenses. Cyber defenses in general, and MT techniques specifically, do not provide a “silver bullet,” protecting against every known cyber attack. Therefore, in practice, multiple defenses should be combined to provide adequate protection of systems. Understanding the impact of these defenses on each other, as well as the composability challenges arising from these defenses, is an open research area. Other important areas for further study are determining if a defense will improve, co-exist, or conflict with another defense and investigating how a defense is influenced by second-order effects, such as an attacker’s reactions to the presence of a new MT technique.

Acknowledgments

The authors gratefully thank the following colleagues at Lincoln Laboratory for their contributions and support to the research on moving target techniques: David Bigelow, Kevin Carter, Robert Cunningham, Veer Dedhia, Thomas Hobson, Mark Rabe, James Riordan, Robert Rudd, and Richard Skowyra.

REFERENCES

1. D. Kaufman, *An Analytical Framework for Cyber Security*, Defense Advanced Research Projects Agency, 2011.
2. H. Okhravi, T. Hobson, D. Bigelow, and W. Strelein, “Finding Focus in the Blur of Moving Target Techniques,” *IEEE Security & Privacy*, vol.12, no.2, 2014, pp.16–26.
3. H. Okhravi, A. Comella, E. Robinson, and J. Haines, “Creating a Cyber Moving Target for Critical Infrastructure Applications Using Platform Diversity,” *International Journal of Critical Infrastructure Protection*, vol. 5, no. 1, 2012, pp. 30–39.
4. B. Salamat, A. Gal, T. Jackson, K. Manivannan, G. Wagner, and M. Franz, “Multi-variant Program Execution: Using Multi-core Systems to Defuse Buffer-Overflow Vulnerabilities,” *Proceedings of the IEEE International Conference on Complex, Intelligent and Software Intensive Systems*, 2008, pp. 843–848.
5. H. Okhravi, J. Riordan, and K. Carter, “Quantitative Evaluation of Dynamic Platform Techniques as a Defensive Mechanism,” , pp. 405-425 in *Research in Attacks, Intrusions and Defenses, Lecture Notes in Computer Science*, A. Stavrou, H. Bos, and G. Portokalidis eds. Cham, Switzerland: Springer International Publishing, 2014.
6. PaX Team, “PaX address space layout randomization (ASLR),” 2003, available at <https://pax.grsecurity.net/docs/aslr.txt>.
7. M. Franz, “E Unibus Ploram: Massive-Scale Software Diversity as a Defense Mechanism,” *Proceedings of the 2010 Workshop on New Security Paradigms*, 2010, pp. 7–16.

8. H. Shacham, "The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)," *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007, pp. 552-561.
9. S. Antonatos, P. Akritidis, E.P. Markatos, and K.G. Anagnostakis, "Defending Against Hitlist Worms Using Network Address Space Randomization," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 51, no. 12, 2007, pp. 3471-3490.
10. T. Hobson, H. Okhravi, D. Bigelow, R. Rudd, and W. Streilein, "On the Challenges of Effective Movement," *Proceedings of the First ACM Workshop on Moving Target Defense*, 2014, pp. 41-50.